МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ЦОО ФИСТ

Кафедра Информационные системы

Дисциплина Распределённые информационные системы

**КУРСОВОЙ ПРОЕКТ**

Тема Разработка приложения «Библиотека» с использованием протокола gRPC

Выполнил студент / Х. А. Нгуен /

подпись инициалы, фамилия

Курс \_\_\_\_четвёртый\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Группа\_\_\_ЦИСТбв-41\_\_\_\_\_\_\_\_\_\_\_\_\_

Направление/специальность 09.03.02 Информационные системы и технологии

Руководитель

должность, ученая степень, ученое звание

Кандаулов Валерий Михайлович

фамилия, имя, отчество

Дата сдачи:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Дата защиты:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск

2023 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ЦОО ФИСТ

Кафедра Информационные системы

Дисциплина Распределённые ИС

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

Студенту ЦИСТбв-41 Нгуен Х. А.

группа фамилия, инициалы

**Тема проекта** Разработка приложения «Библиотека» с использованием протокола gRPC

**Срок сдачи законченного проекта** « » 2023 г.

**Содержание пояснительной записки** (перечень подлежащих разработке вопросов)

Введение

Протокол gRPC

Описание логики работы приложения

Заключение

Руководитель / / В.М. Кандаулов /

должность подпись инициалы, фамилия

«\_\_\_\_\_» 2023 г

Студент / /\_Х. А. Нгуен /

подпись инициалы, фамилия

«\_\_\_\_\_» 2023 г

**Содержание**

[Введение 5](#_Toc137779094)

[Протокол gRPC 6](#_Toc137779095)

[Описание логики приложения 8](#_Toc137779096)

[Архитектура приложения 9](#_Toc137779097)

[Реализация приложения 10](#_Toc137779098)

[Серверная часть (Backend) 10](#_Toc137779099)

[Клиентская часть (Frontend) 21](#_Toc137779100)

[Развертывание приложения 30](#_Toc137779101)

[Используемое технологии и ПО 31](#_Toc137779102)

[Заключение 32](#_Toc137779103)

[Список источников 33](#_Toc137779104)

[Приложение А. Исходный код. 34](#_Toc137779105)

[Проект Gateway 34](#_Toc137779106)

[Program.cs 34](#_Toc137779107)

[AuthorController.cs 34](#_Toc137779108)

[BookController.cs 35](#_Toc137779109)

[GatewayOptions.cs 36](#_Toc137779110)

[BookAddReq.cs 36](#_Toc137779111)

[BookUpdReq.cs 37](#_Toc137779112)

[RpcExceptionFilter.cs 37](#_Toc137779113)

[Проект BookApi 37](#_Toc137779114)

[Program.cs 37](#_Toc137779115)

[Utils.cs 38](#_Toc137779116)

[AppDbContext.cs 38](#_Toc137779117)

[AuthorService.cs 38](#_Toc137779118)

[BookService.cs 39](#_Toc137779119)

[Author.cs 41](#_Toc137779120)

[Book.cs 41](#_Toc137779121)

[BookApiConfig.cs 41](#_Toc137779122)

[Проект FileStorage 41](#_Toc137779123)

[Program.cs 41](#_Toc137779124)

[FileEntry.cs 42](#_Toc137779125)

[FileStorageOptions.cs 42](#_Toc137779126)

[AppDbContext.cs 42](#_Toc137779127)

[StorageService.cs 42](#_Toc137779128)

[Проект WebClient 44](#_Toc137779129)

[Компонент BookPage 44](#_Toc137779130)

[Компонент BookList 46](#_Toc137779131)

[Компонент BookEdit 47](#_Toc137779132)

[Компонент BookUploadFile 50](#_Toc137779133)

[Компонент AuthorPage 51](#_Toc137779134)

# Введение

Распределенные системы (distributed computing) — это системы компоненты которых распределены на разных вычислительных устройствах объединенные в сети, общающийся посредством передачи сообщений между собой. Его иногда называют ещё микросервисной архитектурой. Такие системы встречаются повсюду. Оно встречается в работе многопользовательских компьютерных играх, в облачных приложениях таких Microsoft OneDrive, работа торрент трекеров, в блокчейнах, в веб-сервисах, в работе электронной почты и так далее. И также существует довольно много инструментов управления и разработки распределенных систем, способов и протоколов обмена данными.

В истории самой распространённой распределенной системой можно является локальные компьютерные сети такие как технология Ethernet.

Распределенные системы имеет свои достоинства и недостатки:

* Распределение нагрузки между вычислительными системами
* Масштабируемость системы
* Отказоустойчивость

Цель курсовой работы – получение навыков прикладного программирования, изучение современных инструментов разработки, разработать приложение с использованием протокола передачи данных gRPC и HTTP REST.

# Протокол gRPC

Протокол gRPC — это современный высокопроизводительный Remote Procedure Call (RPC) фреймворк с открытым исходным кодом. Для многих языков программирования существует официальные библиотеки для работы с этим протоколом.

Основные сценарии применения:

* общение между распределенными службами написанные на разных языках программирования и использующие разные технологии
* подключение мобильных устройств, браузерных клиентов к внутренним сервисам
* создание эффективных клиентских библиотек

Протокол gRPC использует protocol buffers (protobuf), которая является языком описания интерфейсов (IDL), для обмена сообщения между сервисами, написанный на разных языках.

Пример protobuf файла:

syntax = "proto3";

package greet;

// The greeting service definition.

service Greeter {

  // Sends a greeting

  rpc SayHello (HelloRequest) returns (HelloReply);

}

// The request message containing the user's name.

message HelloRequest {

  string name = 1;

}

// The response message containing the greetings

message HelloReply {

  string message = 1;

}

Листинг 1 – Пример protobuf файла

В файле protobuf определяются сервисы и сообщения. Есть возможность комментирования через двойные косые черты. Сообщение в protobuf можно считать, как класс в других языках программирования. Поле сообщения объявляется в таком виде <тип данных> <имя поля> = <уникальный номер поля>. Уникальный номер нужен для кодирования сообщения. Также поле можно определить, как необязательным. Для этого используется ключевое слово *optional* перед определением поля сообщения. У каждого сервиса есть метод, который может только один параметр сообщения и один возвращаемый объект сообщения. Такой вызов называют одинарный вызов удаленной процедуры (Unary RPC). Чтобы передавать какой-либо список сообщений нужно ключевое слово stream перед сообщением в определении метода сервиса. Пример:

rpc SayHello (HelloRequest) returns (HelloReply);

либо

rpc SayHello (HelloRequest) returns (stream HelloReply);

либо

rpc SayHello (stream HelloRequest) returns (stream HelloReply);

Первый пример называют Server Streaming RPC, второй – Client Streaming RPC, а третий Bidirectional RPC.

Если нужно передать в сообщениях массив примитивных типов или другого типа сообщения, нужно использовать ключевое слово *repeated*.

Файл protobuf можно импортировать такой же файл и использовать их определения.

Из достоинств protocol buffers имеет:

* Малый размер пакетов
* Быстрый парсинг

Из файлов protobuf можно затем сгенерировать в код с помощью официальной утилиты protoc. Она может в код на разных языках программирования. Полный список можно посмотреть на официальном сайте gRPC [6].

# Описание логики приложения

Приложение должно хранить записи книг, авторов и файлов книг, выводить записи в виде списка или таблицы через веб-интерфейс, возможность поиска книг и возможность загружать и скачивать файлы книг.

**Функции приложения**

* Просмотр списка книг
* Добавление, изменение, удаление книг
* Загрузка файлов книг
* Скачивание файлов
* Просмотра списка авторов книг
* Добавление, изменение, удаление авторов книг

Диаграмма вариантов использования:

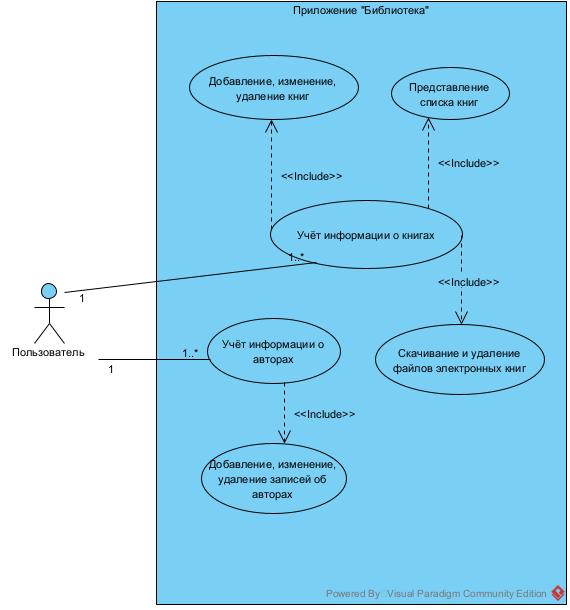


Рисунок 1 – Use Case диаграмма

# Архитектура приложения

Приложение состоит из трёх сервисов:

* BookApi
* FileStorage
* Gateway
* WebClient

Служба BookApi взаимодействует с двумя классами предметной области:

* Book
* Author

Далее схема базы данных BookApi:

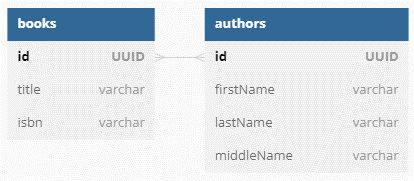


Рисунок 2 – Схема базы данных BookApi

Служба BookApi является внутренним сервисом и должна быть недоступна извне. С ним взаимодействует служба Gateway, которая передаёт запросы от пользователя по протоколу gRPC.

Служба FileStorage хранит файлы книг. Она взаимодействует только с BookApi.

Служба Gateway является основным шлюзом. Через него передаются запросы от пользователя.

Служба WebClient – веб-интерфейс пользователя.

# Реализация приложения

## Серверная часть (Backend)

Для написания сервисов BookApi, FileStorage и Gateway был использован язык C# и фреймворк ASP.NET Core. Для работы сервисов gRPC нужно установить пакет gRPC.AspNetCore. Для работы клиента gRPC нужно ещё установить три пакета - Google.Protobuf, Grpc.Net.Client, Grpc.Tools. Также службы BookApi и FileStorage используют ещё дополнительную библиотеку ORM Entity Framework и SQLite для хранения данных о книгах и авторах. Проекты с заготовленным шаблоном генерировались посредством консольной утилиты dotnet поставляемый с .NET SDK. Структура проекта с сервисами:

└───src

├───BookApi

│ ├───Config

│ ├───Domains

│ ├───Properties

│ └───Services

├───FileStorage

│ ├───Config

│ ├───Models

│ ├───Properties

│ └───Services

├───Gateway

│ ├───Config

│ ├───Controllers

│ ├───Filters

│ └───Properties

└───Proto

Листинг 1 – Структура проекта, полученная с помощью консольной команды tree

В директории src/Proto – хранятся файлы protobuf файлы.

Далее список зависимостей и параметров каждого проекта:

<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>

    <TargetFramework>net7.0</TargetFramework>

    <Nullable>enable</Nullable>

    <ImplicitUsings>enable</ImplicitUsings>

  </PropertyGroup>

  <ItemGroup>

    <PackageReference Include="Google.Protobuf" Version="3.23.2" />

    <PackageReference Include="Grpc.AspNetCore.Server.ClientFactory" Version="2.53.0" />

    <PackageReference Include="Grpc.Net.Client" Version="2.53.0" />

    <PackageReference Include="Grpc.Tools" Version="2.32.0" PrivateAssets="All" />

    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="7.0.5" />

    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.4.0" />

  </ItemGroup>

  <ItemGroup>

    <Protobuf Include="..\Proto\common.proto" Link="Protos\common.proto" />

    <Protobuf Include="..\Proto\file\_storage.proto" Link="Protos\file\_storage.proto" />

    <Protobuf Include="..\Proto\author.proto" GrpcServices="Client" Link="Protos\author.proto" />

    <Protobuf Include="..\Proto\book.proto" GrpcServices="Client" Link="Protos\book.proto" />

  </ItemGroup>

</Project>

Листинг 2 – Файл проекта Gateway

<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>

    <TargetFramework>net7.0</TargetFramework>

    <Nullable>enable</Nullable>

    <ImplicitUsings>enable</ImplicitUsings>

  </PropertyGroup>

  <ItemGroup>

    <Protobuf Include="..\Proto\common.proto" Link="Protos\common.proto" />

    <Protobuf Include="..\Proto\file\_storage.proto" GrpcServices="Server" Link="Protos\file\_storage.proto" />

  </ItemGroup>

  <ItemGroup>

    <PackageReference Include="Grpc.AspNetCore" Version="2.49.0" />

    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="7.0.7" />

  </ItemGroup>

</Project>

Листинг 3 – Файл проекта FileStorage

<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>

    <TargetFramework>net7.0</TargetFramework>

    <Nullable>enable</Nullable>

    <ImplicitUsings>enable</ImplicitUsings>

  </PropertyGroup>

  <ItemGroup>

    <Protobuf Include="..\Proto\common.proto" Link="Protos\common.proto" />

    <Protobuf Include="..\Proto\author.proto" GrpcServices="Server" Link="Protos\author.proto" />

    <Protobuf Include="..\Proto\book.proto" GrpcServices="Server" Link="Protos\book.proto" />

    <Protobuf Include="..\Proto\file\_storage.proto" Link="Protos\file\_storage.proto" />

  </ItemGroup>

  <ItemGroup>

    <PackageReference Include="Grpc.AspNetCore" Version="2.49.0" />

    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="7.0.7" />

  </ItemGroup>

</Project>

Листинг 4 – Файл проекта BookApi

gRPC сервисы в BookApi определяются файлами со следующим текстом:

syntax = "proto3";

import "common.proto";

import "author.proto";

import "file\_storage.proto";

import "google/protobuf/empty.proto";

package books;

service BookService {

    rpc GetBooks(google.protobuf.Empty) returns (stream BookReply);

    rpc AddBook(BookAddRequest) returns (BookReply);

    rpc UpdateBook(BookUpdateRequest) returns (BookReply);

    rpc RemoveBook(common.RemoveRequest) returns (google.protobuf.Empty);

    rpc HasFile(common.HasFileRequest) returns (common.HasFileReply);

    rpc UploadBookFile(stream UploadBookFileChunkRequest) returns (google.protobuf.Empty);

    rpc DownloadBookFile(DownloadBookFileRequest) returns (stream fss.FileChunkMessage);

}

message BookAddRequest {

    string title = 1;

    string isbn = 2;

    repeated string authorIds = 3;

}

message BookUpdateRequest {

    string id = 1;

    string title = 2;

    string isbn = 3;

    repeated string authorIds = 4;

}

message BookRemoveRequest {

    string id = 1;

}

message UploadBookFileChunkRequest {

    string id = 1;

    string filename = 2;

    int32 chunkSize = 3;

    int32 fileSize = 4;

    bytes chunk = 5;

}

message DownloadBookFileRequest {

    string id = 1;

}

message BookReply {

    string id = 1;

    string title = 2;

    string isbn = 3;

    repeated author.AuthorMessage authors = 4;

    bool hasFile = 5;

}

Листинг 5 – Файл book.proto

syntax = "proto3";

import "google/protobuf/empty.proto";

import "common.proto";

package author;

service AuthorService {

    rpc GetAuthors(google.protobuf.Empty) returns (stream AuthorMessage);

    rpc AddAuthor(AuthorAddRequest) returns (AuthorMessage);

    rpc UpdateAuthor(AuthorMessage) returns (AuthorMessage);

    rpc RemoveAuthor(common.RemoveRequest) returns (google.protobuf.Empty);

    rpc SayHello (stream HelloRequest) returns (HelloReply);

    rpc SayHello (HelloRequest) returns (stream HelloReply);

    rpc SayHello (stream HelloRequest) returns (stream HelloReply);

}

message AuthorAddRequest {

    string lastName = 1;

    string firstName = 2;

    string midName = 3;

}

message AuthorMessage {

    string id = 1;

    string lastName = 2;

    string firstName = 3;

    string midName = 4;

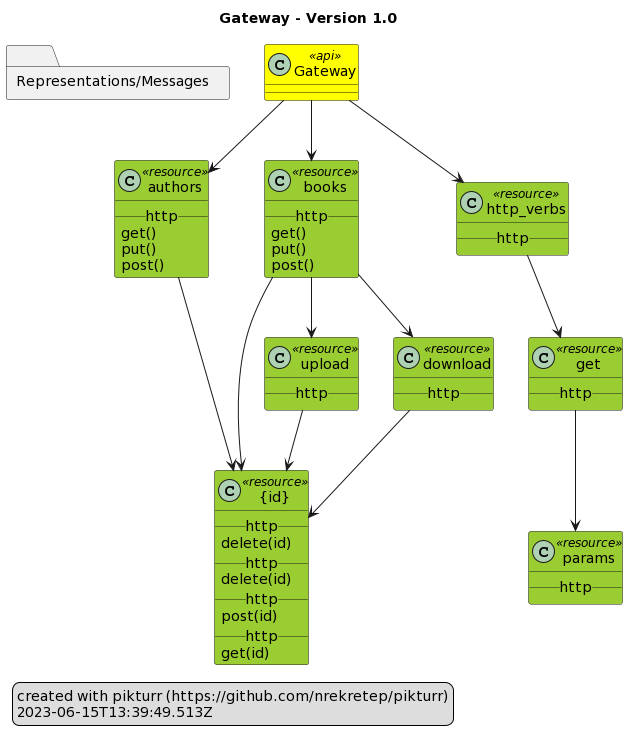
}

Листинг 6 – Файл author.proto

Эти файлы потом генерируются в код на C# с помощью пакета Grpc.Tools при сборке проекта.

Служба Gateway является шлюзом между BookApi и WebClient. Она предоставляет HTTP API для WebClient и при получении запроса от клиента передаёт их BookApi, используя классы BookServiceClient и AuthorServiceClient. Эти классы также автоматически генерируются пакетом Grpc.Tools. Список endpoint-ов Gateway:

* GET /books – список записей книг
* POST /books – создание записи книги
* PUT /books – обновление записи книги
* DELETE /books/{id:Guid} – удаление записи книги
* POST /books/upload/{id:Guid} – загрузка файла книги
* POST /books/download/{id:Guid} – скачивание файла книги
* GET /authors – список записей авторов книг
* POST /authors – добавление нового автора
* PUT /authors – обновление параметров существующего автора
* DELETE /authors/{id:Guid} – удаление автора
* GET / - переадресует по URL клиентского приложения

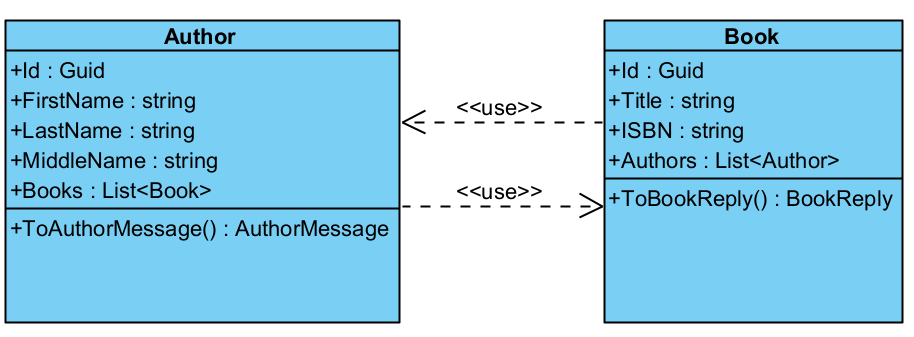
Рисунок 2 – Диаграмма Gateway API

За API отвечают классы контроллеры BookController и AuthorController.

При переходе на страницу /, Gateway перенаправляет на страницу веб-интерфейса. Путь до веб-интерфейса, также задаётся конфигурацией.

Пути URL к другим сервисам находятся в appsettings.json в корневых директориях каждого проекта. Его можно в случае необходимости перезаписать через командную строку, либо через переменные окружения.

Диаграмма классов в BookApi:

Рисунок 3 – Диаграмма классов BookApi

Классы AuthorMessage и BookReply сгенерированы пакетом Grpc.Tools и используются для передачи сообщений между сервисами.

Сервис BookApi общается со службой FileStorage для хранения файлов книг. Если поступает запрос к BookApi на скачивание или загрузку файла книги, то он обращается к нему. Файл file\_storage.proto:

syntax = "proto3";

import "common.proto";

import "google/protobuf/empty.proto";

package fss;

service FileStorageService {

    rpc HasFile(common.HasFileRequest) returns (common.HasFileReply);

    rpc Upload(stream FileChunkMessage) returns (UploadReply);

    rpc Download(DownloadRequest) returns (stream FileChunkMessage);

    rpc DeleteEntry(DeleteEntryRequest) returns (google.protobuf.Empty);

}

message FileChunkMessage {

    int32 chunkSize = 1;

    // Total size of the file

    int32 fileSize = 2;

    string fileName = 3;

    bytes chunk = 4;

}

message UploadReply {

    bool success = 1;

    string uuid = 2;

}

message DownloadRequest {

    string uuid = 1;

}

message DeleteEntryRequest {

    string uuid = 1;

}

Листинг 7 – file\_storage.proto

Загружаемый или скачиваемый файл книги разбивается на коллекцию блоков данных (chunk) и передаётся через gRPC. Сервис FileStorage сохраняет файлы на диск в пути заданный конфигурацией. После загрузки файла FileStorage передаёт обратно сообщение со сгенерированным Guid и потом создаёт объект класса FileEntry. Этот объект затем сохраняется в базу данных. Код класса FileEntry:

namespace FileStorage.Models;

public sealed class FileEntry

{

    public Guid Id { get; set; }

    public string FilePath { get; set; } = string.Empty;

}

Листинг 8 – Класс FileEntry

При удалении записи книги в BookApi, удаляется ещё и запись, вместе с файлом, если существует ещё на диске, в сервисе FileStorage. И вызывается методом DeleteEntry из BookApi.

Просмотреть список endpoint-ов можно на странице, сгенерированные при запуске проекта Gateway по адресу <https://localhost:8443/swagger>.

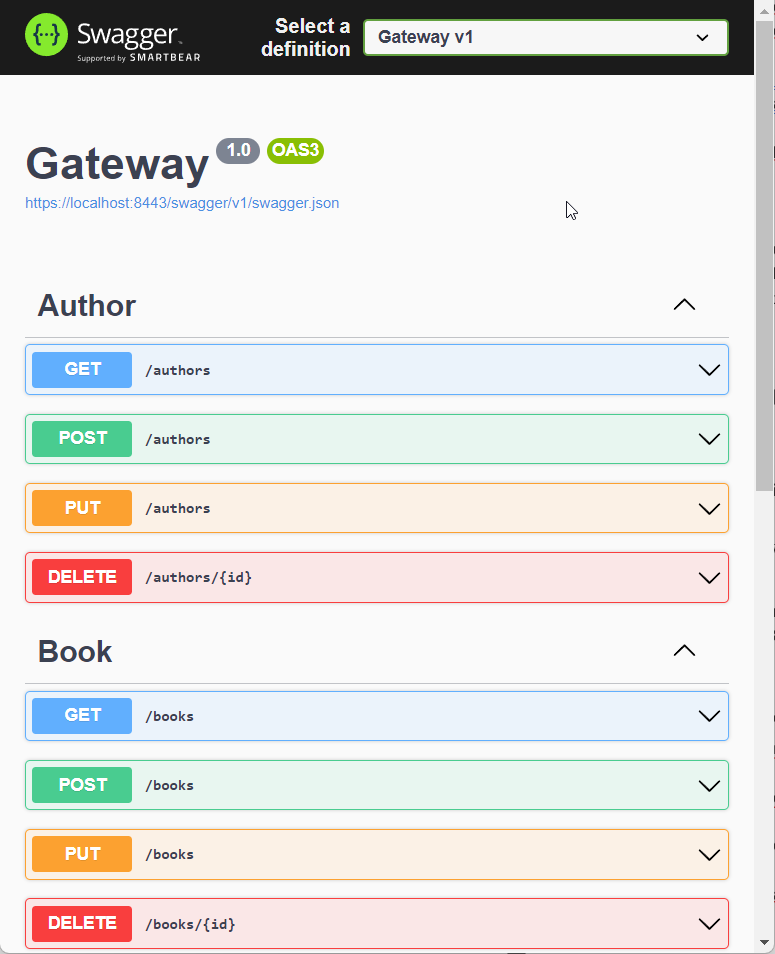


Рисунок 3 – API

Для сервисов определены следующие номера портов:

* 8443 для Gateway
* 12433 для BookApi
* 11433 для FileStorage

## Клиентская часть (Frontend)

Для написания WebClient был использован язык TypeScript и фреймворк Angular 16. Также в зависимостях есть Angular Material, который предоставляет компоненты в стиле Material от Google. Фреймворк использует программный паттерн Dependency Injection для управления зависимостями. Классы, от которых будут зависеть другие компоненты, нужно маркировать атрибутом @Injectable() перед объявления класса. Под компонентами понимается класс представления, имеющий разметку HTML и файл стилей. Их можно задать всё в одном файле или можно разбить на несколько файлов. Компоненты и службы генерируются консольной утилитой *ng*.

Структура проекта:

└───src

├───app

│ ├───components

│ │ ├───book-edit

│ │ ├───book-list

│ │ └───navigation

│ ├───models

│ ├───pages

│ │ ├───author-page

│ │ └───book-page

│ └───services

│ ├───author

│ └───book

├───assets

└───environments

Листинг 6 – Структура проекта WebClient

Путь к API Gateway сервиса находится в файлах environment.\*.ts в директории src/environments. Представлены в виде экспортируемого объекта:

export const environment = {

  api: '192.168.1.52:8443',

};

Листинг 7 – environment.ts

В директории models находятся классы предметной области book.ts и author.ts:

export interface Author {

  id?: string;

  firstName: string;

  lastName: string;

  midName?: string;

}

Листинг 8 – Класс Book

import { Author } from './author';

export interface Book {

  id?: string;

  title: string;

  isbn: string;

  authorIds?: Array<string>;

  authors?: Array<Author>;

  hasFile?: boolean;

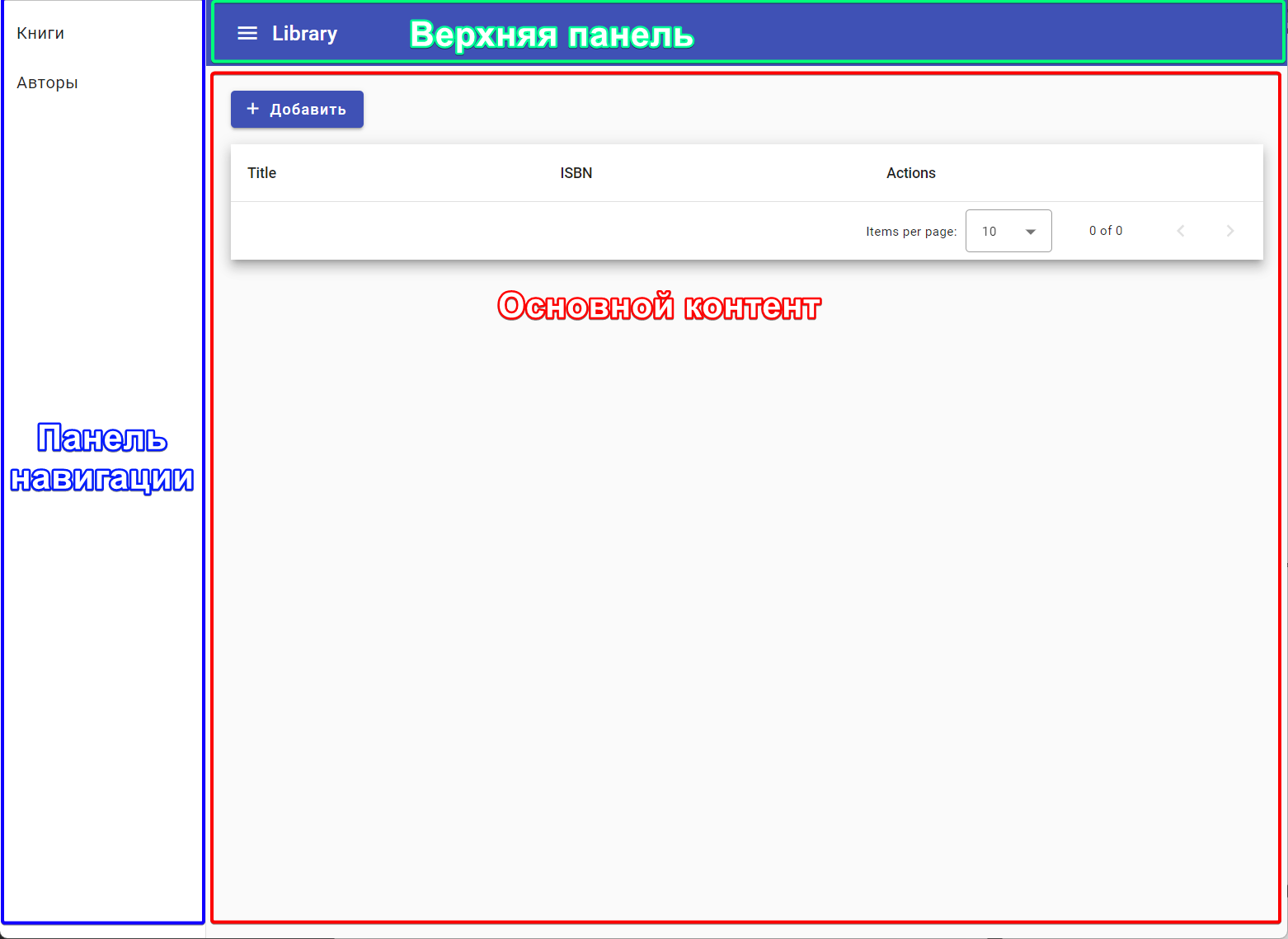
}

Листинг 9 – Класс Author

В папке *src/app/services* находятся сервисы для работы с API. Они используются в компонентах из директории *src/app/pages*.

В проекте используется модуль AppRoutingModule для возможности переходить на другие страницы, набрав адрес в адресной строке браузера.

Разметка страницы выглядит примерно так:

Рисунок 4 – Разметка интерфейса

В панели навигации только есть две ссылки – на основную страницу (ссылки «Книги») и страницу списков авторов (ссылка «Авторы»). Панель навигации всегда находится в закрытом состояний.

В верхней панели отображается кнопка открытия/закрытия панели навигации в виде трёх полосок и название приложения «Library».

На основной странице выводится список книг:

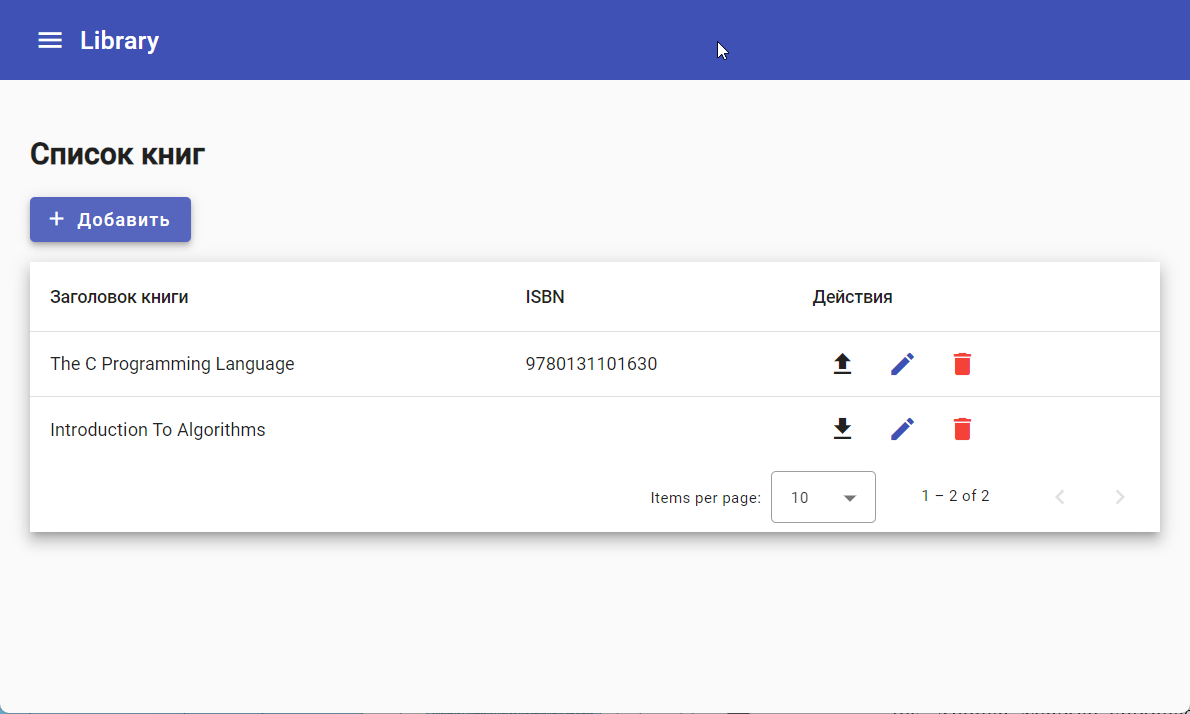


Рисунок 5 – Основная страница

Таблица записей книг имеет следующие поля - «Заголовок книги», «ISBN» (Международный стандартный книжный номер) и «Действия». Также есть возможность сортировки по первым двум полям. Это можно сделать, нажав на ячейку шапку таблицы.

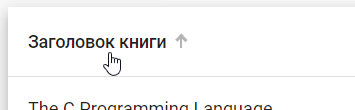


Рисунок 6 – Сортировка по колонкам

У таблицы есть встроенная возможность разбиении строк на страницы (pagination). Количество строк в таблице определена заранее массивом чисел - 10, 20, 50. Она находится под таблицей и имеет следующий вид:

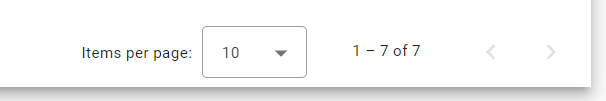


Рисунок 7 - Панель навигации по страницам

Она состоит из выпадающего меню, в котором можно выбрать отображаемое количество элементов на одной странице, потом идёт диапазон отображаемых элементов от общего количества строк в формате <от> - <до> of <общее\_количество\_строк> и кнопки навигации на следующий и предыдущий.

При нажатии кнопки «Добавить» над таблицей, открывается диалоговое окно для создания новой записи:

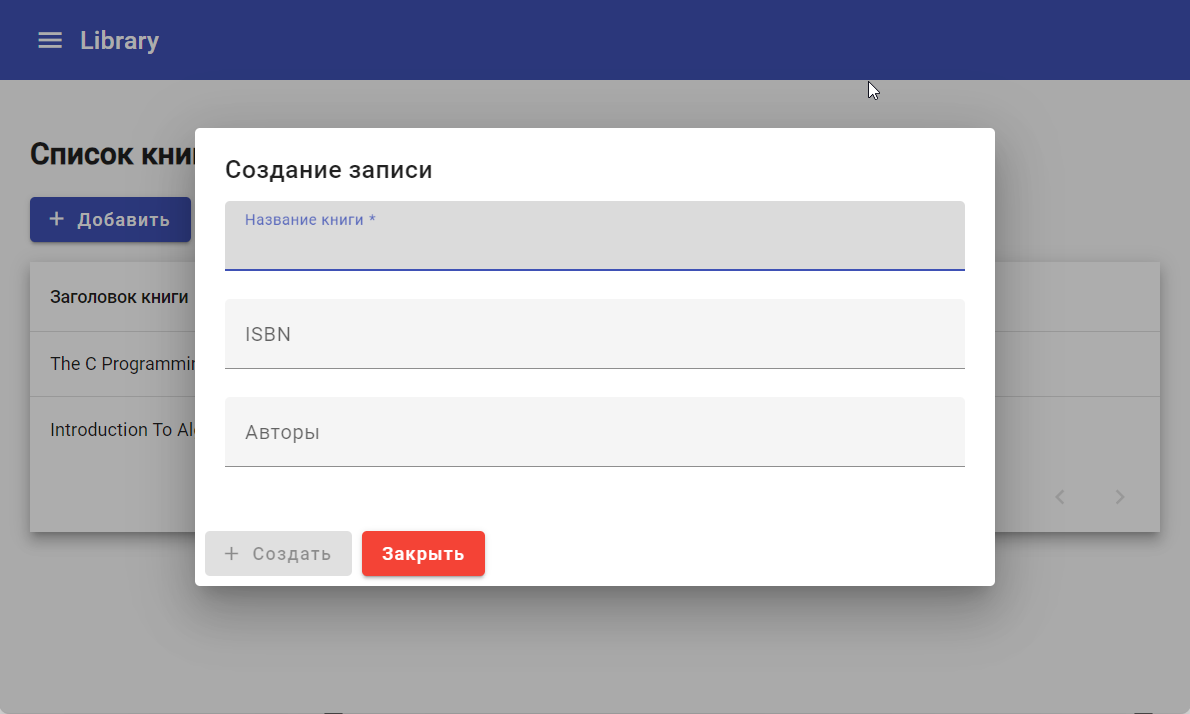
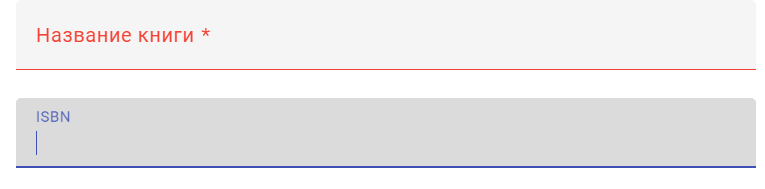


Рисунок 7 – Окно добавления/редактирования записи книги

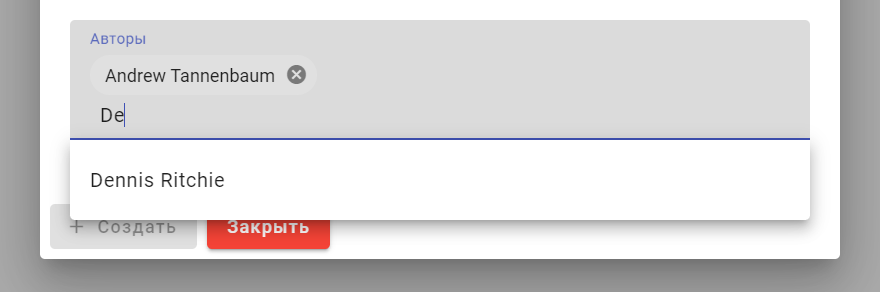
Такое же окно открывается и при редактировании записи нажатием на кнопку с изображением ручки.

Обязательным полем является «Название книги», и она отмечена звёздочкой. Если эта строка будет иметь длину меньше одной символа, то кнопка «Создать»/«Изменить» будет не доступна и не будет окрашена цветом. За валидацию полей в Angular отвечают классы валидаторы Validators.

Если пользователь перейдёт к следующему полю без заполненного обязательного поля, то это цвет контура поля и цвет текста измениться на красный.

Рисунок 8 – Обязательное поле

Авторов книги можно выбирать несколько. Также можно отфильтровать набрав в строке нужные символы. Выбранные элементы отображается в виде чипов (chips).

Рисунок 9 – Поле «Авторы»

Также из следующих функции есть:

* Удаление
* Загрузка файлов книг
* Скачивание файлов книг

У каждой записи книг есть булево поле hasFile, которое говорит есть ли файл книги или нет. Исходя из этого поля клиентское приложение показывает один из двух вариантов действия – скачивание или загрузка.

Рисунок 10 – Кнопка скачивания/загрузки

При нажатии на кнопку загрузки открывается диалоговое окно с выбором файла. Для выбора файла нужно нажать на поле и откроется диалоговое окно для выбора файла с вашего компьютера. Без выбранного файла кнопка загрузить будет недоступна.

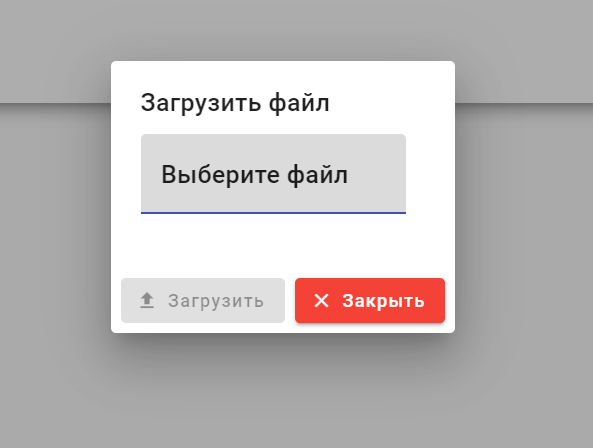


Рисунок 11 – Диалог выбора файла для загрузки

Скачивание файла происходит сразу же при нажатии на кнопку. Для этого используется библиотека file-saver. Она скачивает с API блобы и сохраняет его в виде HTML ссылки и программно нажимает на него и затем открывается диалоговое окно выбора пути для сохранения файла.

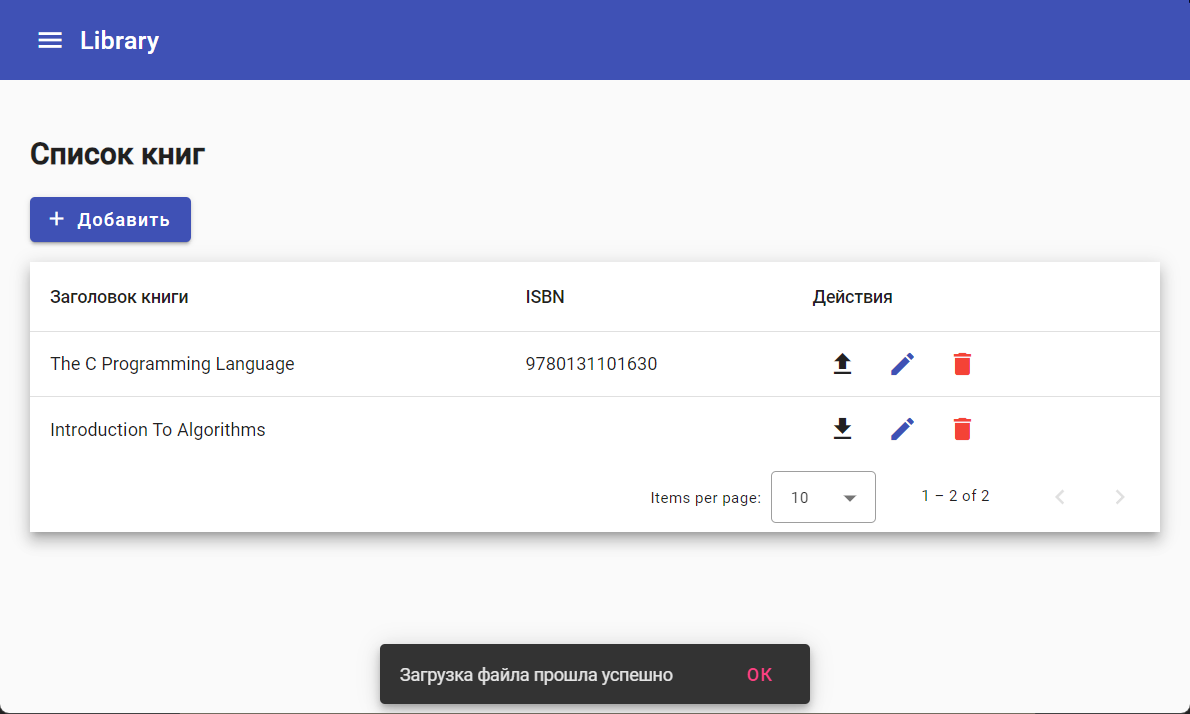
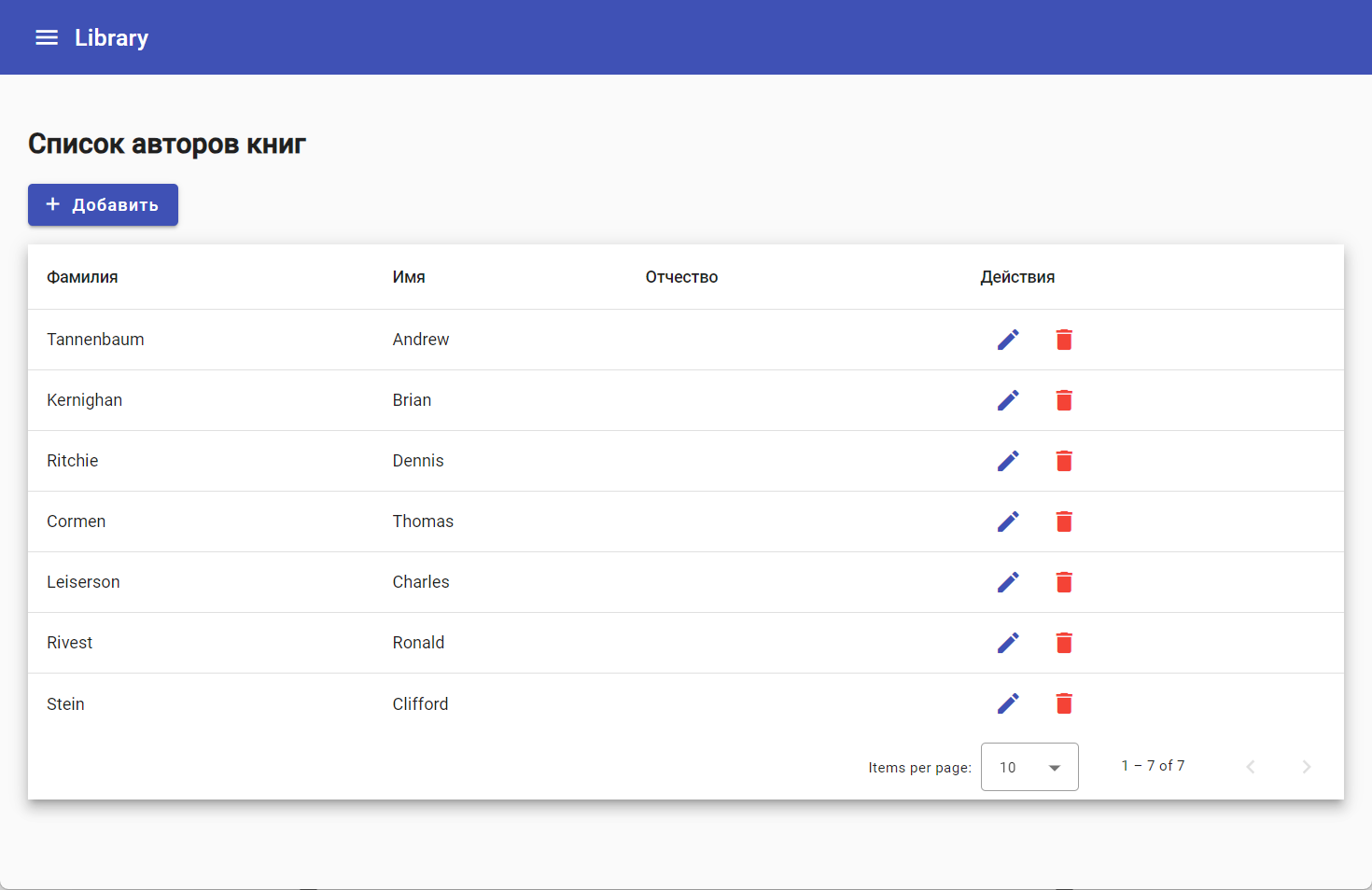


Рисунок 12 – Выпадающее уведомление

Уведомление выпадает об успешности операции над какими-либо записи. Также уведомление выпадает при условии возникновении ошибки при обращениях к серверной части приложения.

Страница со списком авторов книг также представлена в виде таблицы и кнопки добавления новой записи:

Рисунок 13 – Страница «Авторы»

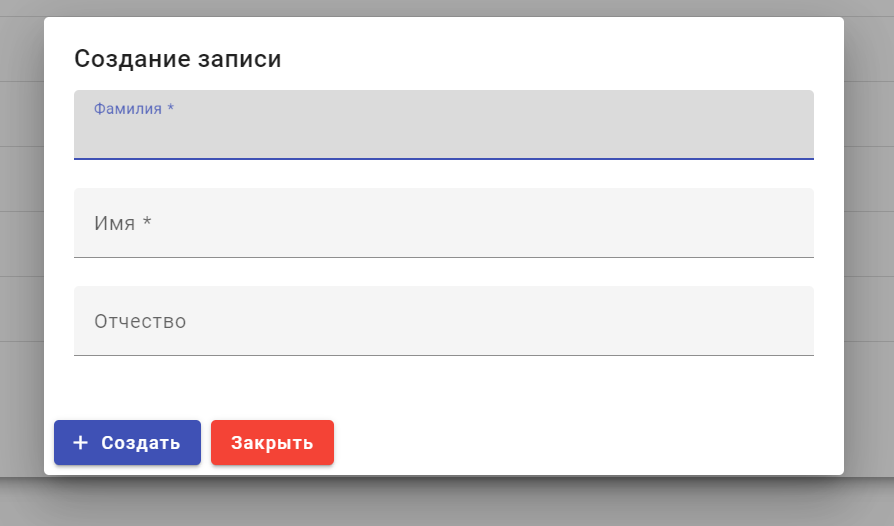
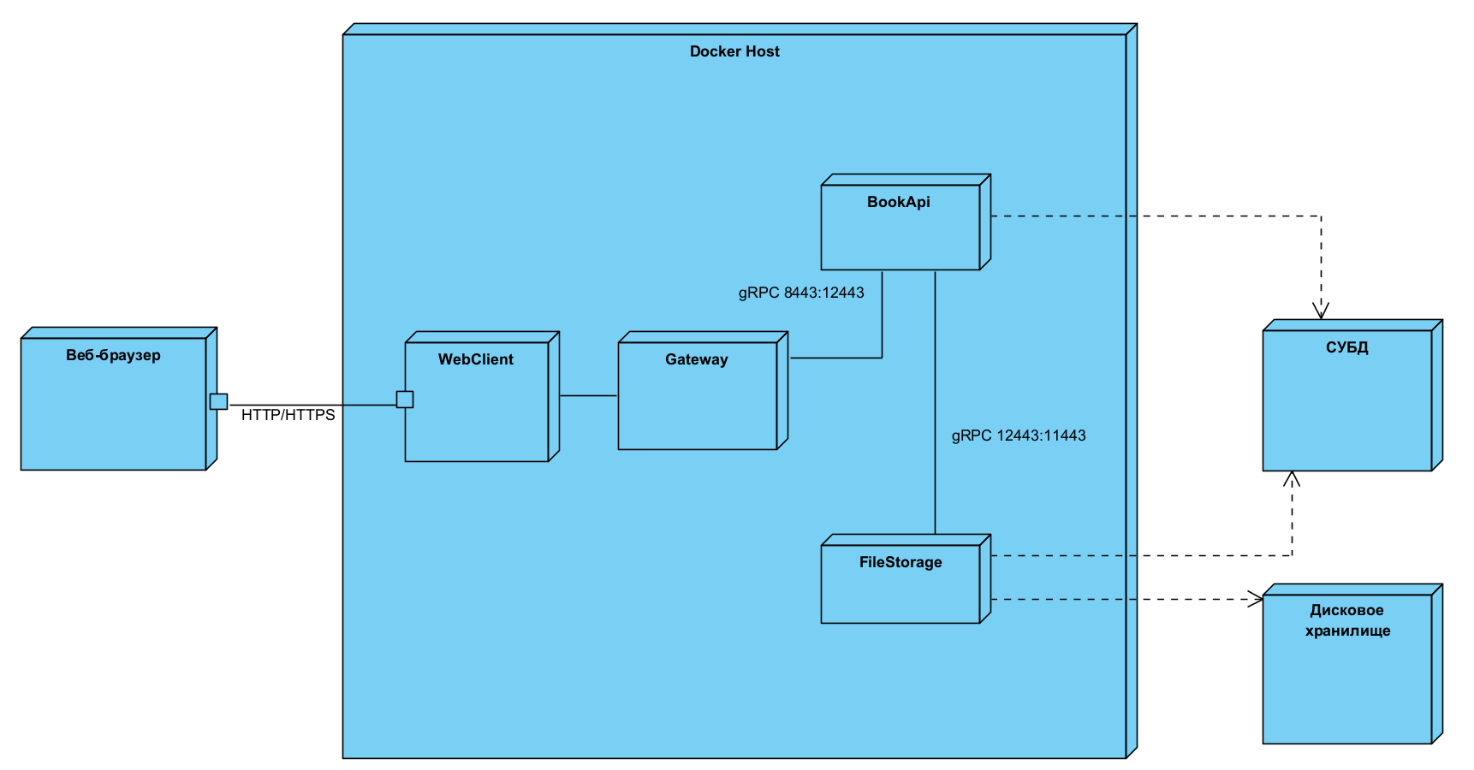


Рисунок 14 – Диалог редактирования записи автора

Поля «Фамилия» и «Имя» являются обязательными для заполнения и заголовки полей отмечены звёздочкой. Эти поля должны иметь не менее 2 двух символов.

# Развертывание приложения

Диаграмма развёртывания приложения с помощью Docker:

Рисунок 15 – Диаграмма развертывания

Для работы служб потребуется среда выполнения .NET Core, которая является мультиплатформенным. Образ со средой выполнения весит примерно от 100 МБ и выше в зависимости от выбранного дистрибутива.

В основном в сервисах используется СУБД SQLite, но если нужно использовать другую СУБД, то нужно поменять код в Program.cs для BookApi и FileStorage и до установить зависимости. К примеру, с SQLite:

builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlite("Data Source=book\_api.db"));

На PostgreSQL:

builder.Services.AddDbContext<AppDbContext>(options => options.UseNpgsql("Host=remote\_host;Database=book\_api;Username=root;Password=root"));

И установить зависимость с помощью утилиты *dotnet*:

dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL

Также потребуется дисковое хранилище для хранения файлов в зависимости от количества книг, которое пользователю нужно сохранить.

# Используемое технологии и ПО

Среда разработки кода – Visual Studio Code.

Языки программирования – C# и TypeScript.

Веб-браузер – Google Chrome 114.0.5735.

Версия .NET – 7.0.302.

Используемые библиотеки и технологии – ASP.NET Core, Entity Framework Core, Grpc.AspNetCore, Angular, Angular Material, file-saver

Программное обеспечение для созданий UML диаграмм – desktop приложение Visual Paradigm Community Edition 17 и plantuml.

Программа для тестирования API – Postman.

# Заключение

В данной курсовой работе было разработано приложение с распределенной архитектурой, которое использует протоколы обмена данными gRPC и HTTP REST. В этой работе было изучено, как работать с этими протоколом, ознакомились с работой этого протокола и как интегрировать gRPC в современном фреймворке для серверных и клиентских програм ASP.NET Core.

# Список источников

1. Marteen Van Steen, Tannenbaum A.S. Distributed Systems. Fourth Edition // ISBN: 978-90-815406-4-3
2. Ajay D. Kshemklayni, Mukesh Singhal. Distributed Computing: Principles, Algorithms, and Systems // Cambridge University Press, 2008. ISBN-13: 978-0-511-39341-9. Электронная версия.
3. ASP.NET Documentation: [Электронный ресурс] // Microsoft, 2023. URL: <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0> .
4. Документация Angular: [Электронный ресурс] // Google LLC, 2023. URL: <https://angular.io/docs> .
5. Документация Angular Material: [Электронный ресурс] // Google LLC, 2023. URL: <https://material.angular.io/components/categories>
6. gRPC Documentation: [Электронный ресурс] // gRPC Authors, 2023. URL: <https://grpc.io/about/>
7. Language Guide (proto 3) | Protocol Buffers Documentation: [Электронный ресурс] // 2023 Google LLC. URL: <https://protobuf.dev/>

# Приложение А. Исходный код.

## Проект Gateway

### Program.cs

var builder = WebApplication.CreateBuilder(args);

var bookApiUrl = builder.Configuration.GetSection("BooksApiUri").Value ?? throw new ArgumentException("The BooksApiUri argument is empty.");

var webClientUri = builder.Configuration.GetSection("WebClientUri").Value ?? throw new ArgumentException("The WebClientUri argument is empty.");

builder.Services.AddControllers(options => {

  options.Filters.Add(typeof(RpcExceptionFilter));

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddGrpcClient<Books.BookService.BookServiceClient>(o =>

{

  o.Address = new Uri(bookApiUrl);

});

builder.Services.AddGrpcClient<Author.AuthorService.AuthorServiceClient>(o =>

{

  o.Address = new Uri(bookApiUrl);

});

builder.Services.AddCors(options => {

  options.AddDefaultPolicy(

  policy =>

  {

    policy.WithOrigins(webClientUri).AllowAnyHeader().AllowAnyMethod().WithExposedHeaders("content-disposition");

  });

});

var app = builder.Build();

if (app.Environment.IsDevelopment()) {

  app.UseSwagger();

  app.UseSwaggerUI();

  app.UseHsts();

}

app.UseCors();

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.MapGet("/", () => { return Results.Redirect(url: webClientUri); });

app.Run();

### AuthorController.cs

namespace Gateway.Controllers;

[EnableCors]

[ApiController]

[Route("/authors")]

public class AuthorController : ControllerBase {

    private readonly AuthorService.AuthorServiceClient \_client;

    public AuthorController(AuthorService.AuthorServiceClient client) {

        \_client = client;

    }

    [HttpGet]

    public async Task<IActionResult> GetAll() {

        var authors = new List<AuthorMessage>();

        using var call = \_client.GetAuthors(new Empty());

        await foreach (var reply in call.ResponseStream.ReadAllAsync())

            authors.Add(reply);

        return Ok(authors);

    }

    [HttpPost]

    public async Task<IActionResult> Create([FromBody] AuthorAddRequest req) => Ok(await \_client.AddAuthorAsync(req));

    [HttpPut]

    public async Task<IActionResult> Update([FromBody] AuthorMessage req) => Ok(await \_client.UpdateAuthorAsync(req));

    [HttpDelete("{id:Guid}")]

    public async Task<IActionResult> Delete([FromRoute] Guid id) {

        await \_client.RemoveAuthorAsync(new Common.RemoveRequest() {

            Id = id.ToString()

        });

        return Ok();

    }

}

### BookController.cs

namespace Gateway.Controllers;

[EnableCors]

[ApiController]

[Route("/books")]

public sealed class BookController : ControllerBase {

  public const int DefaultChunkSize = 1\_048\_576; // 1 MiB

  private readonly BookService.BookServiceClient \_client;

  public BookController(BookService.BookServiceClient client) {

    \_client = client;

  }

  [HttpGet]

  public async Task<IActionResult> GetAll() {

    var books = new List<BookReply>();

    using var call = \_client.GetBooks(new Empty());

    await foreach (var book in call.ResponseStream.ReadAllAsync())

      books.Add(book);

    return Ok(books);

  }

  [HttpPost]

  public async Task<IActionResult> Add([FromBody] Dtos.BookAddReq request) {

    var grpcRequest = request.ToBookAddRequest();

    return Ok(await \_client.AddBookAsync(grpcRequest));

  }

  [HttpPut]

  public async Task<IActionResult> Update([FromBody] Dtos.BookUpdReq request) {

    var grpcRequest = request.ToBookUpdateRequest();

    return Ok(await \_client.UpdateBookAsync(grpcRequest));

  }

  [HttpDelete("{id:Guid}")]

  public async Task<IActionResult> Delete([FromRoute] Guid id) => Ok(await \_client.RemoveBookAsync(new Common.RemoveRequest() { Id = id.ToString() }));

  [HttpGet("hasfile/{id:Guid}")]

  public async Task<IActionResult> HasFile([FromRoute] Guid id) {

    return Ok(await \_client.HasFileAsync(new Common.HasFileRequest() {

      Id = id.ToString()

    }));

  }

  [HttpPost]

  [Route("upload/{id:Guid}")]

  public async Task<IActionResult> Upload([FromRoute] Guid id, [FromForm] IFormFile file) {

    using var stream = file.OpenReadStream();

    using var call = \_client.UploadBookFile();

    var buffer = new byte[DefaultChunkSize];

    int bytesRead = 0;

    while ((bytesRead = stream.Read(buffer, 0, DefaultChunkSize)) > 0) {

      await call.RequestStream.WriteAsync(new UploadBookFileChunkRequest() {

        Id = id.ToString(),

        Filename = file.FileName,

        FileSize = (int)file.Length,

        ChunkSize = DefaultChunkSize,

        Chunk = UnsafeByteOperations.UnsafeWrap(buffer.AsMemory(0, DefaultChunkSize))

      });

    }

    await call.RequestStream.CompleteAsync();

    var reply = await call;

    return Ok(reply);

  }

  [HttpGet]

  [Route("download/{id:Guid}")]

  public async Task<IActionResult> Download([FromRoute] Guid id)

  {

    using var call = \_client.DownloadBookFile(new DownloadBookFileRequest() {

      Id = id.ToString()

    });

    await call.ResponseStream.MoveNext();

    var it = call.ResponseStream.Current;

    var file = new byte[it.FileSize];

    var fileName = it.FileName;

    var offset = 0;

    do {

      it = call.ResponseStream.Current;

      it.Chunk.CopyTo(file, offset);

      offset += it.ChunkSize;

    } while (await call.ResponseStream.MoveNext());

    return File(file, GetMimeType(fileName), fileName);

  }

  private static string GetMimeType(string fileName) => Path.GetExtension(fileName) switch {

    ".pdf" => "application/pdf",

    ".epub" => "application/epub+zip",

    \_ => "application/octet-stream"

  };

}

### GatewayOptions.cs

namespace Gateway.Config;

public class GatewayOptions {

    public const string Gateway = "Gateway";

    public string BookApiUri { get; set; } = string.Empty;

}

### BookAddReq.cs

namespace Gateway.Dtos;

public class BookAddReq {

  public string Title { get; set; } = string.Empty;

  public string? Isbn { get; set; }

  public IEnumerable<string>? AuthorIds { get; set; }

  public BookAddRequest ToBookAddRequest() {

    var grpcRequest = new BookAddRequest() {

      Title = Title,

      Isbn = Isbn,

    };

    grpcRequest.AuthorIds.AddRange(AuthorIds);

    return grpcRequest;

  }

}

### BookUpdReq.cs

namespace Gateway.Dtos;

public class BookUpdReq : BookAddReq {

  public Guid Id { get; set; }

  public BookUpdateRequest ToBookUpdateRequest() {

    var grpcRequest = new BookUpdateRequest() {

      Id = Id.ToString(),

      Title = Title,

      Isbn = Isbn,

    };

    grpcRequest.AuthorIds.AddRange(AuthorIds);

    return grpcRequest;

  }

}

### RpcExceptionFilter.cs

namespace Gateway.Filters;

public class RpcExceptionFilter : IExceptionFilter {

  private static int GetStatusCodeByRpcException(RpcException e) {

    return e.StatusCode switch {

      StatusCode.NotFound => (int)HttpStatusCode.BadRequest,

      StatusCode.InvalidArgument => (int)HttpStatusCode.BadRequest,

      \_ => (int)HttpStatusCode.InternalServerError,

    };

  }

  public void OnException(ExceptionContext context) {

    if (context.Exception is RpcException e) {

      context.Result = new ObjectResult(e.Message) {

        StatusCode = GetStatusCodeByRpcException(e),

      };

    }

  }

}

## Проект BookApi

### Program.cs

var builder = WebApplication.CreateBuilder(args);

var fileStorageUri = builder.Configuration.GetSection("FileStorageUri").Value ?? throw new ArgumentException("The FileStorageUrl argument is empty");

builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlite("Data Source=book\_api.db"));

builder.Services.AddGrpc();

builder.Services.AddGrpcClient<Fss.FileStorageService.FileStorageServiceClient>(o =>

{

    o.Address = new Uri(fileStorageUri);

});

var app = builder.Build();

var scope = app.Services.CreateScope();

var dbContext = scope.ServiceProvider.GetRequiredService<AppDbContext>();

if (dbContext.Database.EnsureCreated())

{

    dbContext.Database.Migrate();

}

app.MapGrpcService<BookService>();

app.MapGrpcService<AuthorService>();

app.Run();

### Utils.cs

namespace BookApi;

public static class Utils {

  public static async Task<TEntity> GetItemByStringGuid<TEntity>(DbContext context, string id) where TEntity : class {

    if (!Guid.TryParse(id, out Guid guid))

      throw new RpcException(new Status(StatusCode.InvalidArgument, $"The provided id = {id} is not valid Guid value."));

    var found = await context.Set<TEntity>().FindAsync(guid)

        ?? throw new RpcException(new Status(StatusCode.NotFound, $"The {nameof(TEntity)} with id = {id} not found."));

    return found;

  }

}

### AppDbContext.cs

namespace BookApi;

public class AppDbContext : DbContext {

    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

    public DbSet<Domains.Author> Authors => Set<Domains.Author>();

    public DbSet<Domains.Book> Books => Set<Domains.Book>();

}

### AuthorService.cs

namespace BookApi.Services;

public sealed class AuthorService : Author.AuthorService.AuthorServiceBase {

  private readonly AppDbContext \_dbContext;

  public AuthorService(AppDbContext context) {

    \_dbContext = context;

  }

  public override async Task GetAuthors(Empty request, IServerStreamWriter<AuthorMessage> responseStream, ServerCallContext context)

  {

    await foreach (var a in \_dbContext.Authors.AsAsyncEnumerable())

      await responseStream.WriteAsync(new AuthorMessage()

      {

        Id = a.Id.ToString(),

        FirstName = a.FirstName,

        LastName = a.LastName,

        MidName = a.MiddleName

      });

  }

  public override async Task<AuthorMessage> AddAuthor(AuthorAddRequest request, ServerCallContext context)

  {

    var newAuthor = new Domains.Author() {

      FirstName = request.FirstName,

      LastName = request.LastName,

      MiddleName = request.MidName,

    };

    await \_dbContext.Authors.AddAsync(newAuthor);

    await \_dbContext.SaveChangesAsync();

    return newAuthor.ToAuthorMessage();

  }

  public override async Task<AuthorMessage> UpdateAuthor(AuthorMessage request, ServerCallContext context) {

    var found = await Utils.GetItemByStringGuid<Domains.Author>(\_dbContext, request.Id);

    found.FirstName = request.FirstName;

    found.LastName = request.LastName;

    found.MiddleName = request.MidName;

    await \_dbContext.SaveChangesAsync();

    return request;

  }

  public override async Task<Empty> RemoveAuthor(RemoveRequest request, ServerCallContext context) {

    var found = await Utils.GetItemByStringGuid<Domains.Author>(\_dbContext, request.Id);

    \_dbContext.Remove(found);

    await \_dbContext.SaveChangesAsync();

    return new Empty();

  }

}

### BookService.cs

namespace BookApi.Services;

public class BookService : Books.BookService.BookServiceBase {

  private readonly AppDbContext \_dbContext;

  private readonly FileStorageService.FileStorageServiceClient \_fssClient;

  public BookService(AppDbContext context, FileStorageService.FileStorageServiceClient fssClient) {

    \_dbContext = context;

    \_fssClient = fssClient;

  }

  public override async Task GetBooks(Empty request, IServerStreamWriter<BookReply> responseStream, ServerCallContext context) {

    await foreach (var book in \_dbContext.Books.Include(b => b.Authors).AsAsyncEnumerable())

      await responseStream.WriteAsync(book.ToBookReply());

  }

  public override async Task<BookReply> AddBook(BookAddRequest request, ServerCallContext context) {

    var newBook = new Book() {

      Title = request.Title,

      ISBN = request.Isbn,

    };

    var authorIds = request.AuthorIds.Distinct().Select(id => id.ToUpper());

    var authors = \_dbContext.Authors.Where((a) => authorIds.Contains(a.Id.ToString()));

    newBook.Authors = await authors.ToListAsync();

    \_dbContext.Books.Add(newBook);

    await \_dbContext.SaveChangesAsync();

    var response = new BookReply() {

      Id = newBook.Id.ToString(),

      Title = newBook.Title,

      Isbn = newBook.ISBN,

      HasFile = false,

    };

    response.Authors.AddRange(authors.Select(a => a.ToAuthorMessage()));

    return response;

  }

  public override async Task<BookReply> UpdateBook(BookUpdateRequest request, ServerCallContext context) {

    var found = await Utils.GetItemByStringGuid<Book>(\_dbContext, request.Id);

    await \_dbContext.Entry(found).Collection(b => b.Authors).LoadAsync();

    var authorIds = request.AuthorIds.Distinct().Select(id => id.ToUpper());

    found.Title = request.Title;

    found.ISBN = request.Isbn;

    found.Authors.Clear();

    var authors = \_dbContext.Authors.Where((a) => authorIds.Contains(a.Id.ToString()));

    found.Authors = await authors.ToListAsync();

    await \_dbContext.SaveChangesAsync();

    return found.ToBookReply();

  }

  public override async Task<Empty> RemoveBook(RemoveRequest request, ServerCallContext context){

    var found = await Utils.GetItemByStringGuid<Book>(\_dbContext, request.Id);

    if (found.FileStorageId != Guid.Empty) {

      await \_fssClient.DeleteEntryAsync(new DeleteEntryRequest() {

        Uuid = found.FileStorageId.ToString(),

      });

    }

    \_dbContext.Books.Remove(found);

    await \_dbContext.SaveChangesAsync();

    return new Empty();

  }

  public override async Task<HasFileReply> HasFile(HasFileRequest request, ServerCallContext context) {

    var found = await Utils.GetItemByStringGuid<Book>(\_dbContext, request.Id);

    return new HasFileReply() {

      File = found.FileStorageId == Guid.Empty

    };

  }

  public override async Task<Empty> UploadBookFile(IAsyncStreamReader<UploadBookFileChunkRequest> requestStream, ServerCallContext context) {

    await requestStream.MoveNext();

    var it = requestStream.Current;

    var found = await Utils.GetItemByStringGuid<Book>(\_dbContext, it.Id);

    using var call = \_fssClient.Upload();

    do {

      it = requestStream.Current;

      await call.RequestStream.WriteAsync(new FileChunkMessage() {

        ChunkSize = it.ChunkSize,

        FileSize = it.FileSize,

        FileName = it.Filename,

        Chunk = it.Chunk,

      });

    } while (await requestStream.MoveNext());

    await call.RequestStream.CompleteAsync();

    var reply = await call.ResponseAsync;

    found.FileStorageId = Guid.Parse(reply.Uuid);

    await \_dbContext.SaveChangesAsync();

    return new Empty();

  }

  public override async Task DownloadBookFile(DownloadBookFileRequest request, IServerStreamWriter<FileChunkMessage> responseStream, ServerCallContext context) {

    var found = await Utils.GetItemByStringGuid<Book>(\_dbContext, request.Id);

    using var call = \_fssClient.Download(new DownloadRequest(){

      Uuid = found.FileStorageId.ToString(),

    });

    await foreach (var message in call.ResponseStream.ReadAllAsync())

      await responseStream.WriteAsync(message);

  }

}

### Author.cs

namespace BookApi.Domains;

public sealed class Author {

    public Guid Id { get; set; }

    public required string FirstName { get; set; }

    public required string LastName { get; set; }

    public string? MiddleName { get; set; }

    public List<Book> Books { get; set; } = new();

    public AuthorMessage ToAuthorMessage() {

        return new AuthorMessage() {

            Id = Id.ToString(),

            FirstName = FirstName,

            LastName = LastName,

            MidName = MiddleName,

        };

    }

}

### Book.cs

namespace BookApi.Domains;

public sealed class Book

{

  public Guid Id { get; set; }

  public required string Title { get; set; }

  public string? ISBN { get; set; }

  public Guid FileStorageId { get; set; } = Guid.Empty;

  public List<Author> Authors { get; set; } = new();

  public BookReply ToBookReply() {

    var reply = new BookReply() {

      Id = Id.ToString(),

      Title = Title,

      Isbn = ISBN,

      HasFile = FileStorageId != Guid.Empty

    };

    reply.Authors.AddRange(Authors.Select(a => a.ToAuthorMessage()));

    return reply;

  }

}

### BookApiConfig.cs

namespace BookApi.Config;

public sealed class BookApiOptions {

  public const string BookApi = "BookApi";

  public string FileStorageUrl { get; set; } = string.Empty;

}

## Проект FileStorage

### Program.cs

var builder = WebApplication.CreateBuilder(args);

builder.Services.Configure<FileStorageOptions>(builder.Configuration.GetSection(FileStorageOptions.FileStorage));

builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlite("Data Source=saved\_files.db"));

builder.Services.AddGrpc();

var app = builder.Build();

var scope = app.Services.CreateScope();

var dbContext = scope.ServiceProvider.GetRequiredService<AppDbContext>();

if (dbContext.Database.EnsureCreated()) {

  dbContext.Database.Migrate();

}

app.MapGrpcService<StorageService>();

app.Run();

### FileEntry.cs

namespace FileStorage.Models;

public sealed class FileEntry {

    public Guid Id { get; set; }

    public string FilePath { get; set; } = string.Empty;

}

### FileStorageOptions.cs

namespace FileStorage.Config;

public class FileStorageOptions {

    public const string FileStorage = "FileStorage";

    public string StoragePath { get; set; } = string.Empty;

    public long MaxFileSize { get; set; } = 104\_857\_600;

}

### AppDbContext.cs

namespace FileStorage;

public sealed class AppDbContext : DbContext

{

  public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

  public DbSet<FileEntry> FileEntries => Set<FileEntry>();

}

### StorageService.cs

namespace FileStorage.Services;

public sealed class StorageService : FileStorageService.FileStorageServiceBase {

  private const int DefaultBufferSize = 1048576;

  private readonly ILogger<StorageService> \_logger;

  private readonly AppDbContext \_dbContext;

  private readonly FileStorageOptions \_options;

  public StorageService(ILogger<StorageService> logger, IOptions<FileStorageOptions> options, AppDbContext dbContext) {

    \_logger = logger;

    \_options = options.Value;

    \_dbContext = dbContext;

  }

  public override async Task<UploadReply> Upload(IAsyncStreamReader<FileChunkMessage> requestStream, ServerCallContext context) {

    Guid uuid;

    try {

      await requestStream.MoveNext();

      var it = requestStream.Current;

      if (string.IsNullOrEmpty(it.FileName) || it.FileSize <= 0)

        throw new ArgumentException("The filename or filesize is not provided.");

      uuid = Guid.NewGuid();

      var date = DateTimeOffset.Now;

      var fileExtension = Path.GetExtension(it.FileName);

      var fileNameWithoutExt = Path.GetFileNameWithoutExtension(it.FileName).Trim();

      if (fileNameWithoutExt.EndsWith('.'))

        fileNameWithoutExt.Remove(fileNameWithoutExt.Length - 1);

      if (fileNameWithoutExt.Length > 100)

        fileNameWithoutExt = fileNameWithoutExt[..100];

      var fileName = $"[{date.Year}-{date.Month}-{date.Day}] ({uuid}) {fileNameWithoutExt}.{fileExtension}";

      var filePath = Path.Combine(\_options.StoragePath, fileName);

      await using var writeStream = File.Create(filePath);

      do {

        it = requestStream.Current;

        await writeStream.WriteAsync(it.Chunk.Memory);

      } while (await requestStream.MoveNext());

      await \_dbContext.FileEntries.AddAsync(new Models.FileEntry(){

        Id = uuid,

        FilePath = filePath,

      });

      await \_dbContext.SaveChangesAsync();

    } catch (Exception e) {

      \_logger.LogError(e, "Upload error");

      return new UploadReply() { Success = false };

    }

    return new UploadReply() {

      Success = true,

      Uuid = uuid.ToString() };

  }

  public override async Task Download(DownloadRequest request, IServerStreamWriter<FileChunkMessage> responseStream, ServerCallContext context) {

    if (!Guid.TryParse(request.Uuid, out Guid id))

      throw new RpcException(new Status(StatusCode.NotFound, $"The provided id = {request.Uuid} is not Guid."));

    var found = await \_dbContext.FileEntries.FindAsync(id)

        ?? throw new RpcException(new Status(StatusCode.NotFound, $"File with id = {id} not found"));

    await using var readStream = File.OpenRead(found.FilePath);

    var fileInfo = new FileInfo(found.FilePath);

    var fileName = Path.GetFileName(found.FilePath);

    var buffer = new byte[DefaultBufferSize];

    int bytesRead = 0;

    while ((bytesRead = await readStream.ReadAsync(buffer.AsMemory(0, DefaultBufferSize))) > 0) {

      await responseStream.WriteAsync(new FileChunkMessage() {

        FileSize = (int)fileInfo.Length,

        FileName = fileName,

        ChunkSize = bytesRead,

        Chunk = UnsafeByteOperations.UnsafeWrap(buffer.AsMemory(0, bytesRead))

      });

    }

  }

  public override async Task<Empty> DeleteEntry(DeleteEntryRequest request, ServerCallContext context) {

    if (!Guid.TryParse(request.Uuid, out Guid id))

      throw new RpcException(new Status(StatusCode.InvalidArgument, "The provided uuid is not Guid."));

    var found = await \_dbContext.FileEntries.FindAsync(id)

                ?? throw new RpcException(new Status(StatusCode.NotFound, $"The file entry with id = {id} not found."));

    if (File.Exists(found.FilePath))

      File.Delete(found.FilePath);

    \_dbContext.Remove(found);

    await \_dbContext.SaveChangesAsync();

    return new Empty();

  }

}

## Проект WebClient

### Компонент BookPage

<h2>Список книг</h2>

<div class="add-button-container">

  <button mat-raised-button color="primary" (click)="onAdd()">

    <mat-icon>add</mat-icon>

    <span>Добавить</span>

  </button>

</div>

<app-book-list

  [dataSource]="dataSource"

  (download)="onDownload($event)"

  (upload)="onUpload($event)"

  (edit)="onEdit($event)"

  (delete)="onDelete($event)"

></app-book-list>

export class BookPageComponent implements OnInit {

  public dataSource = new BookListDataSource();

  private \_books = new Array<Book>();

  private \_authors = new Array<Author>();

  constructor(

    private \_bookService: BookService,

    private \_authorService: AuthorService,

    private \_snackBar: MatSnackBar,

    private \_dialog: MatDialog

  ) {}

  public ngOnInit(): void {

    this.\_bookService

      .getBooks()

      .pipe(

        catchError((err) => {

          this.\_snackBar.open('Ошибка!', 'OK');

          console.error(err);

          return throwError(() => err);

        })

      )

      .subscribe((books) => {

        this.\_books = books;

        this.updateDataSource();

      });

    this.\_authorService

      .getAuthors()

      .pipe(

        catchError((err) => {

          this.\_snackBar.open('Ошибка!', 'OK');

          console.error(err);

          return throwError(() => err);

        })

      )

      .subscribe((authors) => {

        this.\_authors = authors;

      });

  }

  public updateDataSource(): void {

    this.dataSource.setData(this.\_books);

  }

  public onAdd(): void {

    const dialogRef = this.\_dialog.open(BookEditComponent, { data: { authors: this.\_authors } });

    dialogRef.afterClosed().subscribe((result) => {

      if (result && result.book) {

        if (result.book.authors) {

          result.book.authorIds = result.book.authors.map((a) => a.id || '');

        }

        this.\_bookService

          .addBook(result.book)

          .pipe(

            catchError((err) => {

              this.\_snackBar.open('Ошибка!', 'OK');

              console.error(err);

              return throwError(() => err);

            })

          )

          .subscribe((book) => {

            this.\_books.push(book);

            this.updateDataSource();

            this.\_snackBar.open('Запись добавлена', 'ОК');

          });

      }

    });

  }

  public onEdit(book: Book): void {

    const dialogRef = this.\_dialog.open(BookEditComponent, { data: { book, authors: this.\_authors } });

    dialogRef.afterClosed().subscribe((result) => {

      if (result && result.book) {

        if (result.book.authors) {

          result.book.authorIds = result.book.authors.map((a) => a.id || '');

        }

        this.\_bookService

          .updateBook(result.book)

          .pipe(

            catchError((err) => {

              this.\_snackBar.open('Ошибка!', 'OK');

              console.error(err);

              return throwError(() => err);

            })

          )

          .subscribe((b) => {

            const foundIndex = this.\_books.findIndex(

              (book) => book.id === b.id

            );

            if (foundIndex > 0) {

              this.\_books[foundIndex] = b;

              this.updateDataSource();

              this.\_snackBar.open('Запись обновлена', 'ОК');

            }

          });

      }

    });

  }

  public onDelete(id: string): void {

    this.\_bookService

      .removeBook(id)

      .pipe(

        catchError((err) => {

          this.\_snackBar.open('Ошибка!', 'OK');

          console.error(err);

          return throwError(() => err);

        })

      ).subscribe(() => {

        this.\_books = this.\_books.filter((b) => b.id !== id);

        this.updateDataSource();

        this.\_snackBar.open('Запись успешно удалена', 'ОК');

      });

  }

  public onUpload(id: string): void {

    const dialogRef = this.\_dialog.open(UploadBookFileComponent, { data: { id } });

    dialogRef.afterClosed().subscribe((response) => {

        this.\_bookService.uploadBook(id, response.file)

          .pipe(

            catchError((err) => {

              this.\_snackBar.open('Ошибка!', 'OK');

              console.error(err);

              return throwError(() => err);

            })

          )

          .subscribe(() => {

            const found = this.\_books.find((b) => b.id === id);

            if (found)

              found.hasFile = true;

            this.updateDataSource();

            this.\_snackBar.open('Загрузка файла прошла успешно', 'ОК');

          });

    });

  }

  public onDownload(id: string): void {

    this.\_bookService.downloadBook(id).subscribe((response) => {

      let filename = '';

      const disposition = response.headers.get('Content-Disposition');

      if (disposition && disposition.indexOf('attachment') !== -1) {

        const filenameRegex = /filename[^;=\n]\*=((['"]).\*?\2|[^;\n]\*)/;

        const matches = filenameRegex.exec(disposition);

        if (matches != null && matches[1]) filename = matches[1].replace(/['"]/g, '');

      }

      if (response.body) FileSaver.saveAs(response.body, filename);

    });

  }

}

### Компонент BookList

  <table mat-table class="full-width-table" matSort aria-label="Elements">

    <ng-container matColumnDef="title">

      <th mat-header-cell \*matHeaderCellDef mat-sort-header>Заголовок книги</th>

      <td mat-cell \*matCellDef="let row">{{ row.title }}</td>

    </ng-container>

    <ng-container matColumnDef="isbn">

      <th mat-header-cell \*matHeaderCellDef mat-sort-header>ISBN</th>

      <td mat-cell \*matCellDef="let row">{{ row.isbn }}</td>

    </ng-container>

    <ng-container matColumnDef="actions">

      <th mat-header-cell \*matHeaderCellDef mat-sort-header>Действия</th>

      <td mat-cell \*matCellDef="let row">

        <span \*ngIf="row.hasFile; else elseNoFile">

          <button mat-icon-button (click)="onDownloadClick(row.id)">

            <mat-icon>download</mat-icon>

          </button>

        </span>

        <ng-template #elseNoFile>

          <button mat-icon-button (click)="onUploadClick(row.id)">

            <mat-icon>upload</mat-icon>

          </button>

        </ng-template>

        <button mat-icon-button color="primary" (click)="onEditClick(row)">

          <mat-icon>edit</mat-icon>

        </button>

        <button mat-icon-button color="warn" (click)="onDeleteClick(row.id)">

          <mat-icon>delete</mat-icon>

        </button>

      </td>

    </ng-container>

    <tr mat-header-row \*matHeaderRowDef="displayedColumns"></tr>

    <tr mat-row \*matRowDef="let row; columns: displayedColumns"></tr>

  </table>

  <mat-paginator #paginator [length]="dataSource?.dataLength" [pageIndex]="0" [pageSize]="10" [pageSizeOptions]="[10, 50, 100]"></mat-paginator>

export class BookListComponent implements AfterViewInit {

  @ViewChild(MatPaginator) public paginator!: MatPaginator;

  @ViewChild(MatSort) public sort!: MatSort;

  @ViewChild(MatTable) public table!: MatTable<BookListItem>;

  @Input() public dataSource?: BookListDataSource;

  @Output() public edit = new EventEmitter<Book>();

  @Output() public delete = new EventEmitter<string>();

  @Output() public upload = new EventEmitter<string>();

  @Output() public download = new EventEmitter<string>();

  public onEditClick(book: Book): void { this.edit.emit(book); }

  public onDeleteClick(id: string): void { this.delete.emit(id); }

  public onUploadClick(id: string): void { this.upload.emit(id); }

  public onDownloadClick(id: string): void { this.download.emit(id); }

  public displayedColumns = ['title', 'isbn', 'actions'];

  constructor() {}

  public ngAfterViewInit(): void {

    if (this.dataSource) {

      this.dataSource.sort = this.sort;

      this.dataSource.paginator = this.paginator;

      this.table.dataSource = this.dataSource;

    }

  }

}

### Компонент BookEdit

<h2 mat-dialog-title>

  <span \*ngIf="!dialogData.book; else editTitle">Создание записи</span>

  <ng-template #editTitle>Изменение записи</ng-template>

</h2>

<form mat-dialog-content class="editForm" [formGroup]="bookForm">

  <mat-form-field>

    <mat-label> Название книги </mat-label>

    <input matInput formControlName="title" />

  </mat-form-field>

  <mat-form-field>

    <mat-label> ISBN </mat-label> <input matInput formControlName="isbn" />

  </mat-form-field>

  <mat-form-field>

    <mat-label> Авторы </mat-label>

    <mat-chip-grid #chipGrid>

      <mat-chip-row

        \*ngFor="let author of selectedOptions"

        (removed)="removeAuthor(author)"

      >

        {{ displayFn(author) }}

        <button matChipRemove>

          <mat-icon>cancel</mat-icon>

        </button>

      </mat-chip-row>

    </mat-chip-grid>

    <input

      matInput

      formControlName="authors"

      #authorInput

      [formControl]="authorsFormCtrl"

      [matChipInputFor]="chipGrid"

      [matAutocomplete]="auto"

      [matChipInputSeparatorKeyCodes]="separatorKeysCodes"

      (matChipInputTokenEnd)="addAuthor($event)"

    />

    <mat-autocomplete auto="matAutocomplete" (optionSelected)="selectedAuthor($event)" >

      <mat-option \*ngFor="let option of filteredOptions | async" [value]="option" >

        {{ displayFn(option) }}

      </mat-option>

    </mat-autocomplete>

  </mat-form-field>

</form>

<div mat-dialog-actions>

  <button

    mat-raised-button

    color="primary"

    (click)="onEdit()"

    [disabled]="!bookForm.valid"

  >

    <mat-icon \*ngIf="!dialogData.book; else editBtnIcon">add</mat-icon>

    <ng-template #editBtnIcon><mat-icon>edit</mat-icon></ng-template>

    <span \*ngIf="!dialogData.book; else editBtnTitle"> Создать </span>

    <ng-template #editBtnTitle>Изменить</ng-template>

  </button>

  <button mat-raised-button color="warn" (click)="onClose()">Закрыть</button>

</div>

export interface BookEditDialogData {

  book?: Book;

  authors: Author[];

}

@Component({

  selector: 'app-book-edit',

  templateUrl: './book-edit.component.html',

  styleUrls: ['./book-edit.component.scss'],

})

export class BookEditComponent implements OnInit {

  public titleFormCtrl: FormControl<string | null> = new FormControl('', [

    Validators.required,

    Validators.minLength(1),

  ]);

  public isbnFormCtrl: FormControl<string | null> = new FormControl('');

  public authorsFormCtrl: FormControl<string | null> = new FormControl('');

  public separatorKeysCodes: number[] = [ENTER, COMMA];

  public bookForm = new FormGroup({

    title: this.titleFormCtrl,

    isbn: this.isbnFormCtrl,

    authors: this.authorsFormCtrl,

  });

  public filteredOptions?: Observable<Author[]>;

  public selectedOptions: Author[] = [];

  @ViewChild('authorInput') authorInput!: ElementRef<HTMLInputElement>;

  constructor(

    private \_dialogRef: MatDialogRef<BookEditComponent, BookEditDialogData>,

    @Inject(MAT\_DIALOG\_DATA) public dialogData: BookEditDialogData

  ) {}

  public ngOnInit(): void {

    if (this.dialogData.book) {

      this.bookForm.setValue({

        title: this.dialogData.book.title,

        isbn: this.dialogData.book.isbn,

        authors: null,

      });

      if (this.dialogData.book.authors) this.selectedOptions.push(...this.dialogData.book.authors);

    }

    this.filteredOptions = this.bookForm.get('authors')?.valueChanges.pipe(

      startWith(null),

      map((value) => {

        return (typeof value === 'string') ? this.\_filter(value as string) : this.\_getAllAuthors().slice();

      })

    );

  }

  public displayFn = (author: Author) => author.firstName + ' ' + author.lastName;

  public selectedAuthor(event: MatAutocompleteSelectedEvent) {

    this.selectedOptions.push(event.option.value);

    this.authorInput.nativeElement.value = '';

    this.authorsFormCtrl.setValue(null);

  }

  public addAuthor(event: MatChipInputEvent) {

    const value = event.value ?? undefined;

    if (value) this.selectedOptions.push();

    event.chipInput!.clear();

    this.authorsFormCtrl.setValue(null);

  }

  public removeAuthor(author: Author) {

    const index = this.selectedOptions.indexOf(author);

    if (index >= 0)

      this.selectedOptions.splice(index, 1);

  }

  public onEdit() {

    if (this.bookForm.valid) {

      this.\_dialogRef.close({

        book: {

          id: this.dialogData.book?.id,

          title: this.titleFormCtrl.value!,

          isbn: this.isbnFormCtrl.value ?? '',

          authors: this.selectedOptions,

        },

        authors: [],

      });

    }

  }

  public onClose() { this.\_dialogRef.close(undefined); }

  private \_filter(name: string): Author[] {

    const filterValue = name.toLowerCase();

    return this.\_getAllAuthors().filter(

      (option) =>

        option.firstName.toLowerCase().includes(filterValue) ||

        option.lastName.toLowerCase().includes(filterValue) ||

        option.midName?.toLowerCase().includes(filterValue)

    );

  }

  private \_getAllAuthors(): Author[] {

    const selectedAuthorIds = this.selectedOptions.map(

      (selected) => selected.id

    );

    return this.dialogData.authors.filter(

      (a) => !selectedAuthorIds.includes(a.id)

    );

  }

}

### Компонент BookUploadFile

<h2 mat-dialog-title>Загрузить файл</h2>

<div mat-dialog-content>

  <mat-form-field>

    <mat-toolbar>

      <input matInput [value]="fileName" readonly />

    </mat-toolbar>

    <input type="file" id="fileInput" (change)="selectFile($event)" name="fileInput"/>

  </mat-form-field>

</div>

<div mat-dialog-actions>

  <button mat-raised-button color="primary" (click)="onUploadClick()" [disabled]="!currentFile">

    <mat-icon>upload</mat-icon>

    <span>Загрузить</span>

  </button>

  <button mat-raised-button color="warn" (click)="onCloseClick()">

    <mat-icon>close</mat-icon>

    <span>Закрыть</span>

  </button>

</div>

export interface UploadBookFileData { id: string; }

export interface UploadBookFileResponse { file: File; }

@Component({

  selector: 'app-upload-book-file',

  templateUrl: './upload-book-file.component.html',

  styleUrls: ['./upload-book-file.component.scss'],

})

export class UploadBookFileComponent {

  public fileName: string = 'Выберите файл';

  public currentFile?: File;

  constructor(private \_dialogRef: MatDialogRef<UploadBookFileComponent, UploadBookFileResponse>, @Inject(MAT\_DIALOG\_DATA) public data: UploadBookFileData) {}

  public selectFile(event: Event) {

    if (

      event.target instanceof HTMLInputElement &&

      event.target.files &&

      event.target.files[0]

    ) {

      const file = event.target.files[0];

      this.currentFile = file;

      this.fileName = this.currentFile.name;

    } else {

      this.fileName = 'Select File';

    }

  }

  public onUploadClick() {

    if (this.currentFile) {

      this.\_dialogRef.close({

        file: this.currentFile,

      });

    }

  }

  public onCloseClick() {

    this.\_dialogRef.close(undefined);

  }

}

### Компонент AuthorPage

<h2>Список авторов книг</h2> <div class="add-button-container"><button mat-raised-button color="primary" (click)="onAdd()"> <mat-icon>add</mat-icon> <span>Добавить</span> </button></div> <app-author-list [dataSource]="dataSource" (edit)="onEdit($event)" (delete)="onDelete($event)"></app-author-list>

@Component({

  selector: 'app-author-page',

  templateUrl: './author-page.component.html',

  styleUrls: ['./author-page.component.scss'],

})

export class AuthorPageComponent implements OnInit {

  public dataSource = new AuthorListDataSource();

  private \_authors = new Array<Author>();

  constructor( private \_service: AuthorService, private \_snackBar: MatSnackBar, public dialog: MatDialog) {}

  public ngOnInit(): void {

    this.\_service

      .getAuthors()

      .pipe(

        catchError((err) => {

          this.\_snackBar.open('Ошибка!', 'OK');

          console.error(err);

          return throwError(() => err);

        })

      )

      .subscribe((authors) => {

        this.\_authors = authors;

        this.updateDataSource();

      });

  }

  public onAdd(): void {

    const dialogRef = this.dialog.open(AuthorEditComponent, { data: {} });

    dialogRef.afterClosed().subscribe((result) => {

      if (result && result.author) {

        this.\_service

          .addAuthor(result.author)

          .pipe(

            catchError((err) => {

              this.\_snackBar.open('Ошибка!', 'OK');

              console.error(err);

              return throwError(() => err);

            })

          )

          .subscribe((a) => {

            this.\_authors.push(a);

            this.updateDataSource();

            this.\_snackBar.open('Запись добавлена', 'ОК');

          });

      }

    });

  }

  public onEdit(author: Author): void {

    const dialogRef = this.dialog.open(AuthorEditComponent, { data: { author } });

    dialogRef.afterClosed().subscribe((result) => {

      if (result && result.author) {

        this.\_service.updateAuthor(result.author).subscribe((a) => {

          const foundIndex = this.\_authors.findIndex(

            (author) => author.id === a.id

          );

          if (foundIndex > 0) {

            this.\_authors[foundIndex] = a;

            this.updateDataSource();

            this.\_snackBar.open('Запись обновлена', 'ОК');

          }

        });

      }

    });

  }

  public onDelete(id: string): void {

    this.\_service.removeAuthor(id).pipe(

        catchError((err) => {

          this.\_snackBar.open('Ошибка!', 'OK');

          console.error(err);

          return throwError(() => err);

        })

      )

      .subscribe(() => {

        this.\_authors = this.\_authors.filter((b) => b.id !== id);

        this.updateDataSource();

        this.\_snackBar.open('Запись успешно удалена', 'ОК');

      });

  }

  public updateDataSource(): void {

    this.dataSource.setData(this.\_authors);

  }

}